User Manual: Model Checking

Abstract:

The Model Checking User Interface (MCUI) for AutoFOCUS allows to specify and verify formal properties over AutoFOCUS models. MCUI allows to define many parameters and settings for the model checkers, the translation of the models and the generation of the counter examples. NuSMV and Cadence SMV can be used for checking.

The Model Checking User Interface has been developed by Validas AG, and bases on the model checking integration of TU Munich.

Contents:

1 Introduction 3

2 Installation 4

2.1 NuSMV 4
2.2 Cadence SMV 5

3 Models and Properties 6

3.1 Supported Models 6
3.2 Supported Properties 7
3.3 Property Tags 9
3.4 Bounded Model Checking 10

4 Using MCUI 12

4.1 Example 13

5 Available Settings 16

5.1 Model 16
5.2 Properties 17
5.3 Options 19
5.4 Encoding 20
5.5 MSC 22

6 Known Bugs 25

7 Literature 26

1 Introduction

The Model Checking User Interface (MCUI) controls the model checking process of AutoFOCUS models (see [Slo00], [PS99]). The process consists of the following steps:

-1. Install model checker and define general settings.
0. Build the model using AutoFOCUS.
1. Check if it is finite (model checkable).
2. Define formal properties.
3. Configure specific settings.
4. Execute the check.
5. Examine results (counter examples for falsified properties)

AutoFOCUS model checking (see [PS99], [Wim00]) allows to use CTL, LTL and user defined property patterns for verification. It is restricted to finite models.

The following main features are supported by MCUI:

- Finiteness checking.
- Specification of properties using the model-based property editor.
- Configuration of the following parameters:
    - checker options,
    - encoding options,
    - user defined property patterns and
    - counter example generation settings.

The manual is structured as follows: Chapter 2 describes the installation of the model checkers, Chapter 3 explains the checkable models. Chapter 4 describes the usage of the tool by means of an example. All available features are described in Chapter 5. Chapter 6 contains open issues.

Note that the MCUI is still in an experimental phase, hence neither Validas, nor TU Munich give any warranty for the correctness. The user works at his own risk.

The tool is free, professional support can be requested from Validas AG (mailto: support@validas.de).

2 Installation

The Model Checking User Interface (MCUI) is freely available as part of AutoFOCUS. However using it requires to install a model checker separately.

MCUI is available since AutoFOCUS release 0.9.1. New versions of the MCUI are distributed with new versions of AutoFOCUS. Therefore no additional version numbers are necessary for the MCUI. MCUI currently supports two different model checkers:

1. NuSMV
2. Cadence SMV

It suffices to install one of them.

The connection to AutoFocus is established within MCUI. Here the Checker can be selected and the path to it has to be specified. Note that it is necessary to specify the type of the checker (NuSMV or Cadence SMV).

Figure 1 shows the selection of the model checker. It is part in the Options panel (see Section 5.3).

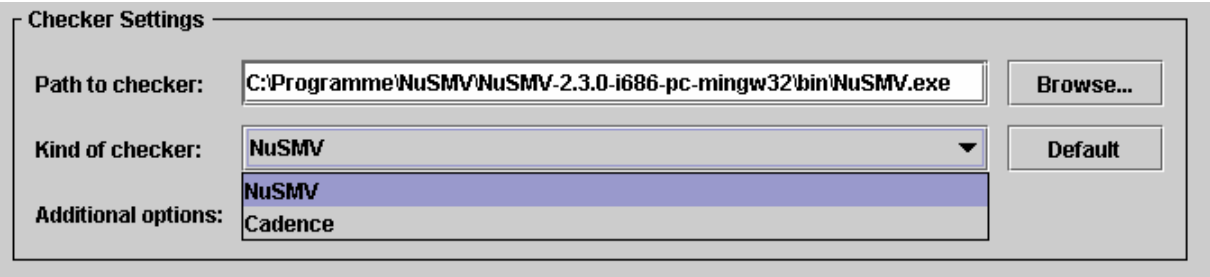

Figure 1: Installation Settings

2.1 NuSMV

NuSMV is a new version of the SMV model checking tool. It is a free tool (with LGPL) and can be used for academic and industrial applications. NuSMV is open source project.

NuSMV can be downloaded from **http://nusmv.irst.itc.it/.** Precompiled versions of NuSMV are available for Linux, Windows and Max OS X systems.

To install NuSMV on your system, follow the above link and download NuSMV (chaff is only needed for bounded model checking).

MCUI has been tested using NuSMV version 2.3 with Linux and Windows systems. Running NuSMV on Windows systems required to add the library libexpat.dll into Windows\System directory. This library has to be downloaded separately, e.g. from http://www.dll-files.com/dllindex/dll-files.shtml?libexpat.

2.2 Cadence SMV

The Cadence model checker comes from Carnegie Mellon University and is now available from Cadence Design Systems. There is a free version for academic and testing purposes and a professional one for commercial and development use.

The free version from Cadence SMV can be downloaded from http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/. Cadence SMV is available on the following platforms: i386/Linux, Windows and Sparc/Solaris.

MCUI has been tested using Cadence SMV release 10-11-02p46.

3 Models and Properties

Model checking is only supported for finite AutoFOCUS models. The properties can be specified with the MCUI. Some features of AutoFOCUS are not supported for model checking. This chapter describes the supported subset of models and the available properties that can be checked. Note that the verifiable models should also be small, otherwise model checking might not terminate due to the exponential complexity of the search space ("state explosion problem").

3.1 Supported Models

In general all consistent, simulable and finite models are supported for model checking, however there are some new features of AutoFOCUS that have not been added into the model checking integration.
The following features of AutoFOCUS are not supported:

- fixed-point types
- type abbreviations (using the keyword type)
- range types
- type implementations,
- polymorphic types, like sets and arrays,
- Float, Double, Char, String,
- recursive types and functions
- multiplication and division.

The subset of checkable models is checked when MCUI is started. If the model does not belong to the subset, an error message is issued, for example in Figure 2, where the **`in`**-operator for sets is used.

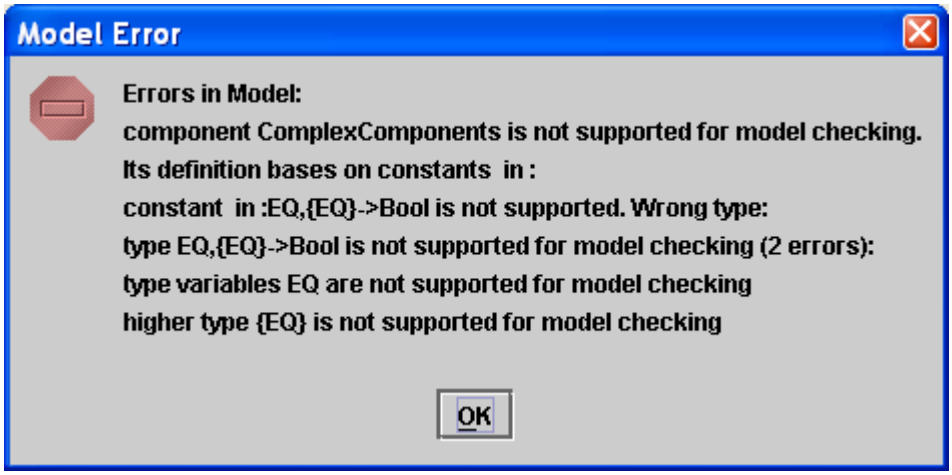
Model Error
Errors in Model:
component ComplexComponents is not supported for model checking.
Its definition bases on constants in :
constant in :EQ,{EQ}->Bool is not supported. Wrong type:
type EQ,{EQ}->Bool is not supported for model checking (2 errors):
type variables EQ are not supported for model checking
higher type {EQ} is not supported for model checking
OK

Figure 2: Message for un-checkable Models

User defined functions, data types, and pattern matching are supported. The following predefined constants and functions are supported for model checking:

- **`True`, `False`,**
- integer numbers,
- **`not`, `if-then-else-fi`**
- `=>` (implication)
- `<=>` (bi-implication)
- `||`, `&&`,
- **`bor`, `band`, `bxor`** (bitwise operations)
- `=`, `!=`, `==`
- `<`, `>`, `<=`, `>=`
- `+`, `–` (binary and unary)

Note that the usually priority rules apply between the infix operators.

3.2 Supported Properties

CTL and LTL properties are supported. The atomic expressions are expressions over the model, and can refer to all elements of the model. To refer to the value of a port `P` that is send using `P!V`, the following "channel-functions" are supported:

- **`Msg(V)`**: denotes that the messages `V` has been send (on `P`)
- **`Val(P)`**: refers to the value of the port `P`
- **`NoVal`** denotes the value of an empty port

- **`is_Msg`** and **`is_NoVal`** can be used to check the ports for presence and absence of values.

Note that these terms are also inserted into the generated counter examples.

The property editor supports the definition of properties using menus for all ports, local variables, states, types and functions of the model. Furthermore property patterns for frequently used properties are available (see Figure 14 on page 18). See Section 5.4 for the possibility to define further property patterns.

The following elements can be used for the formulation of properties:

- Temporal operators of LTL (see [NuSMV], and [CGH97]):
  - `[]`: always (see the example in **Fehler! Verweisquelle konnte nicht gefunden werden.**), corresponds to G (globally) in CTL
  - `<>`:sometimes, corresponds to F (future) in CTL
  - `()`: next, corresponds to X (next) in CTL
  - `|_|`: Until (note this is an infix operator), corresponds to U in CTL
- temporal operators of CTL (see [NuSMV]). Note: CTL operators are preceded by an $ to separate them from the normal functions available in DTDs:
  - **`$AG`**: always in all paths
  - **`$EG`**: always in one path
  - **`$AF`**: sometimes in all paths
  - **`$EF`**: sometimes in one path
  - **`$AX`**: next in all paths
  - **`$EX`**: next in one path
  - **`$AU`**: until in all paths (infix operator)
  - **`$EU`**: until in one path (infix operator)
- All predefined constants and functions (see the last list in Section 3.1)
- All constants and functions defined in the DTD of the model

- References to the following model elements (if necessary with qualified names):
  - Ports
  - Local variable
  - States,
  - `@` (the current state)
- Property Patterns (non executable DTD) can be used to define further properties, for example "fun never(x) = $AG(not(x));" See Section 5.4 for property patterns.

Stating for example that a component **`C`** is never in state **`X`**, when port **`P`** has the value **`True`** is: **`[](is_Msg(C.P) && Val(C.P)==True => C.@!=C.X)`**.

3.3 Property Tags

Since AutoFCUS and AutoFOCUS 2 currently has no integration of the property features (Editor and persistency) there is another mechanism for the persistent specification of properties.

Using property-tags in the comments of components properties can be specified. The following two tags are supported:

- **`{property = <property-term>}`**
- **`{sysproperty = <property-term>}`**

The first tag specifies a property for the component on which the tag is attached, the second tag specifies a property for the top component ("system"), which cannot be edited directly in AutoFOCUS.

Several tags can be defined using ";". This allows to specify many properties using:

- **`{property = <property-term1>; property =<property-term2>}`**

Both tags are evaluated when MCUI starts: the properties are extracted from the components and added to the model, such that they can be edited and verified within the Properties panel of MCUI.

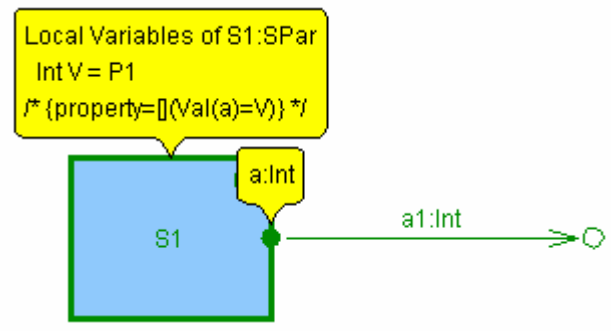

Figure 3: Property Tag on a Component

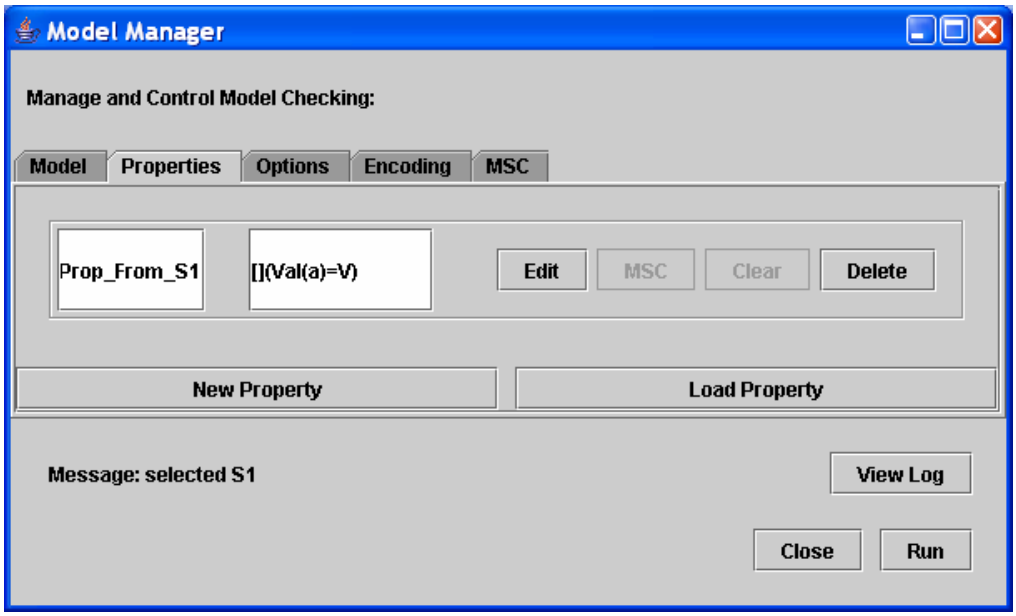

Figure 4: Property from the Property Tag

Figure 3 shows an example of a tagged property, which can be visualized in MCUI in the Properties panel (see Figure 4), if the component `C1` is selected in the Model panel.

3.4 Bounded Model Checking

Bounded Model Checking (BMC) is a restricted from of model checking. BMC verifies a property only for a certain number (bound) of steps. If the property is invalidated by a counter example within the given bound, it is false also in the general case. If no counter example is found within the given bound, the property can be true (if there exists no counter examples), or false (if the existing counter example is longer than the given number of steps). Therefore the colour of properties that are verified with BMC is not green but yellow (see).

BMC can be applied for larger models that cannot be model checked.

BMC can be enabled by a simple switch (in the Options panel, see Section 5.3). If BMC is enabled, the maximal bound has to be entered.

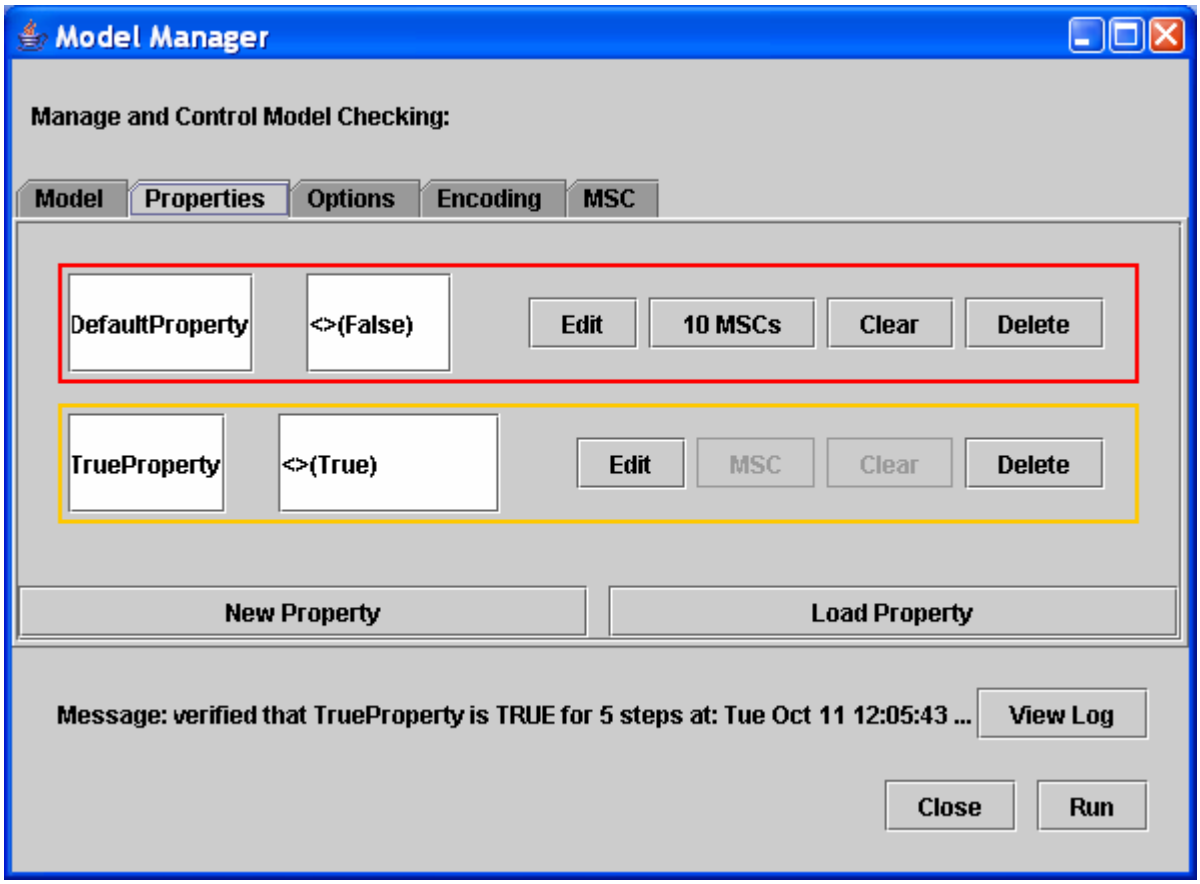

Figure 5: Bounded model checking results

The used model checkers integrate bounded model checkers, or support this feature. Cadence SMV is available with zChaff (which is for research purpose), NuSMV is available with SIM, zChaff and MiniSAT. SIM can be used in any kind of projects. The selection of the used bounded model checker is done using options, when calling the checker, e.g. **`smv -bmc -l <bound> -satsolver zchaff`**. MCUI provides some defaults for these options, that can be changed in the configuration file **`validas.cfg`** (see Section 5.3)

4 Using MCUI

The MCUI is integrated into AutoFOCUS and AutoFOCUS 2 prototype. In AutoFOCUS it can be started from the Project Browser is an project, or a component of the language **AutoMODE** is selected using the menu Project -> Start AutoMODE API -> Model Checking (see Figure 6).

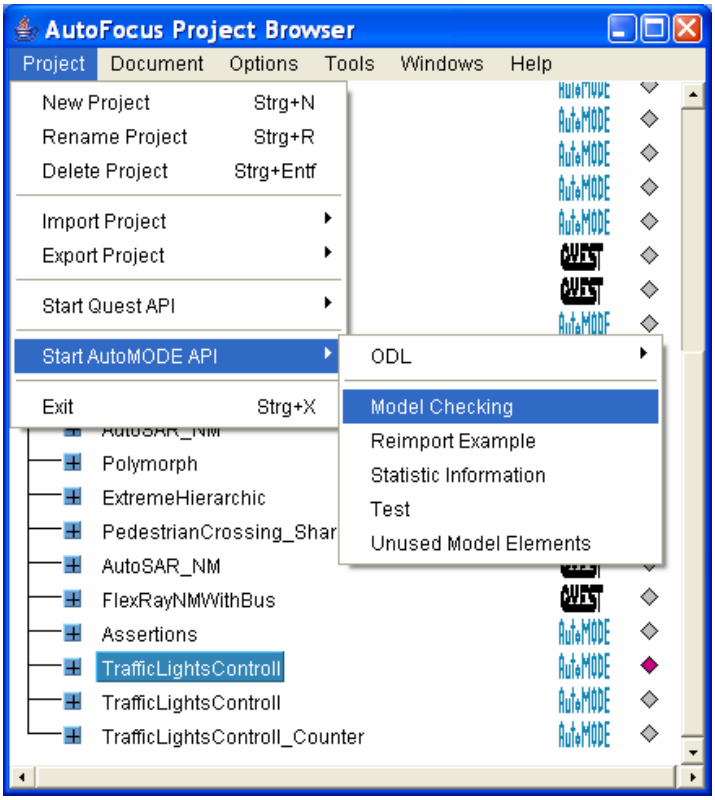

Figure 6: Starting MCUI from AutoFOCUS

After the work with MCUI (when closing MCUI) MCUI will ask if the project (with the generated counter examples) shall be re-imported into the AutoFOCUS repository. If no counter examples have been generated the model will no be re-imported. A re-imported project has the suffix **_Counter** appended to the project name (see last project in Figure 6).

In AutoFOCUS 2 the MCUI can also be started from the browser window, if a component is selected. It can be started by using the Tools menu entry Model Checking, or directly from the browser with the right mouse button menu (see Figure 7).

The model checking can be started by pressing the Run bottom. Messages are displayed in the message part of MCUI and can be viewed in the logging file (by pressing the View Log button).

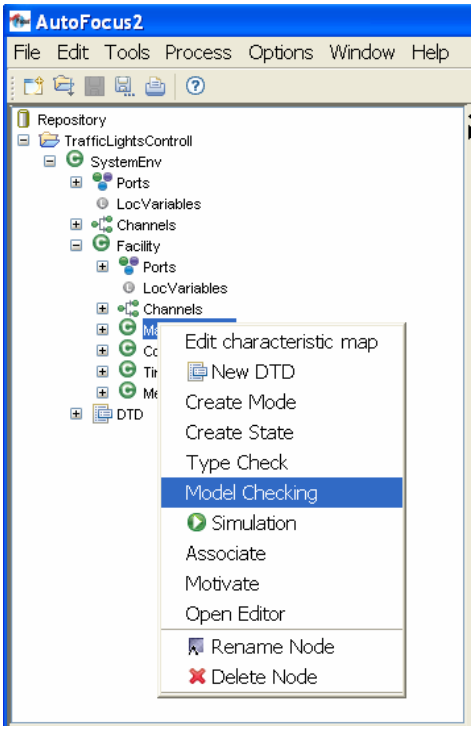

Figure 7: Starting MCUI from AutoFOCUS2

4.1 Example

The model checking of AutoFOCUS is illustrated by the well known example from the modelling tutorial. The structure from the system is depicted in Figure 8. The behaviour of the component **`Controller`** is defined by the STD in Figure 9.

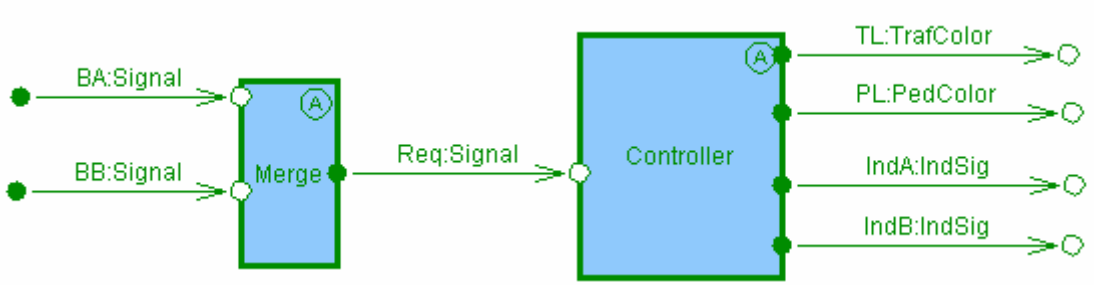

Figure 8: Structure of Traffic-Lights Example

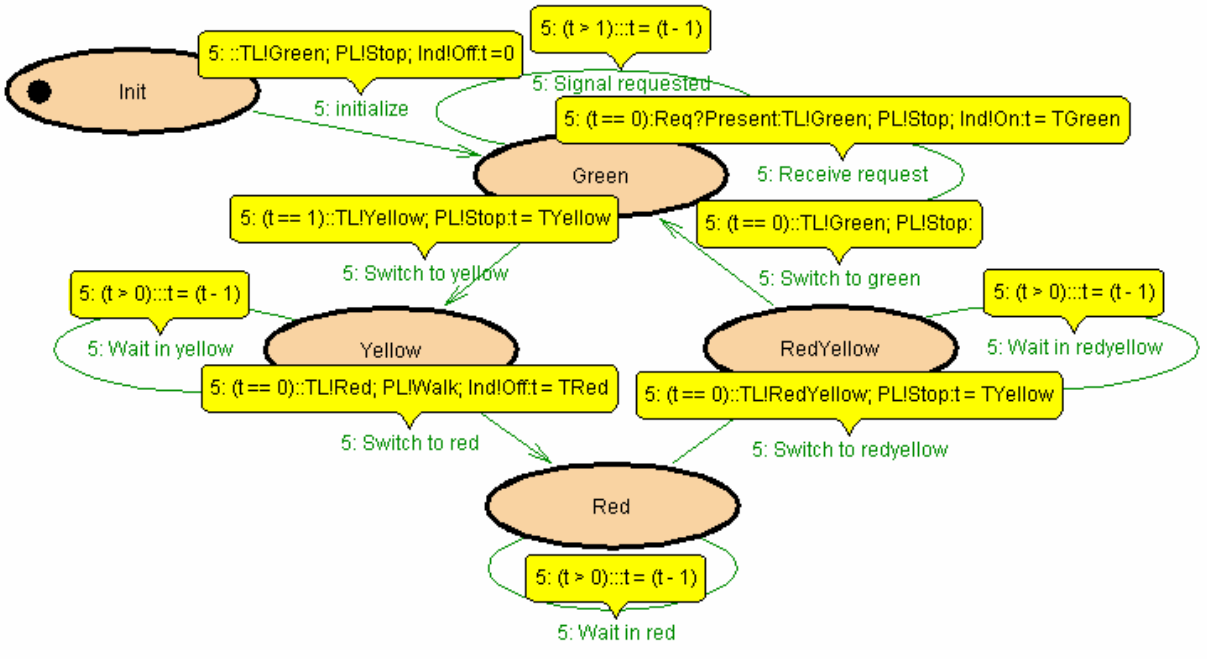
Init
5: ::TL!Green; PL!Stop; Ind!Off:t = 0
5: initialize
5: (t > 1):::t = (t - 1)
5: Signal requested
5: (t == 0):Req?Present:TL!Green; PL!Stop; Ind!On:t = TGreen
Green
5: Receive request
5: (t == 1)::TL!Yellow; PL!Stop:t = TYellow
5: Switch to yellow
5: (t == 0)::TL!Green; PL!Stop:
5: Switch to green
5: (t > 0):::t = (t - 1)
5: Wait in yellow
Yellow
RedYellow
5: (t > 0):::t = (t - 1)
5: Wait in redyellow
5: (t == 0)::TL!Red; PL!Walk; Ind!Off:t = TRed
5: (t == 0)::TL!RedYellow; PL!Stop:t = TYellow
5: Switch to red
5: Switch to redyellow
Red
5: (t > 0):::t = (t - 1)
5: Wait in red

Figure 9: Behavior of `Controller`

The interesting properties of the example that shall be checked are:

1. The cars and the pedestrians have never both green
2. The cars and the pedestrians have never both red
3. The cars never wait forever
4. The pedestrians never wait for ever

The formalization of the properties into temporal logic is:

```
1. never(Val(TL)==Green && Val(PL)==Walk)
2. never(Val(TL)==Red && Val(PL)==Stop)
3. [](Val(TL)==Red => <>(Val(TL)==Green)))
4. [](Val(PL)==Stop => <>(Val(PL)==Walk)))
```

The results of the verification is that property 1 to 3 are valid, while property 4 is invalid (see Figure 10). The counter example to the property (see Figure 11) shows the case that pedestrians that do not submit requests might wait forever, which contradict the formulated property.

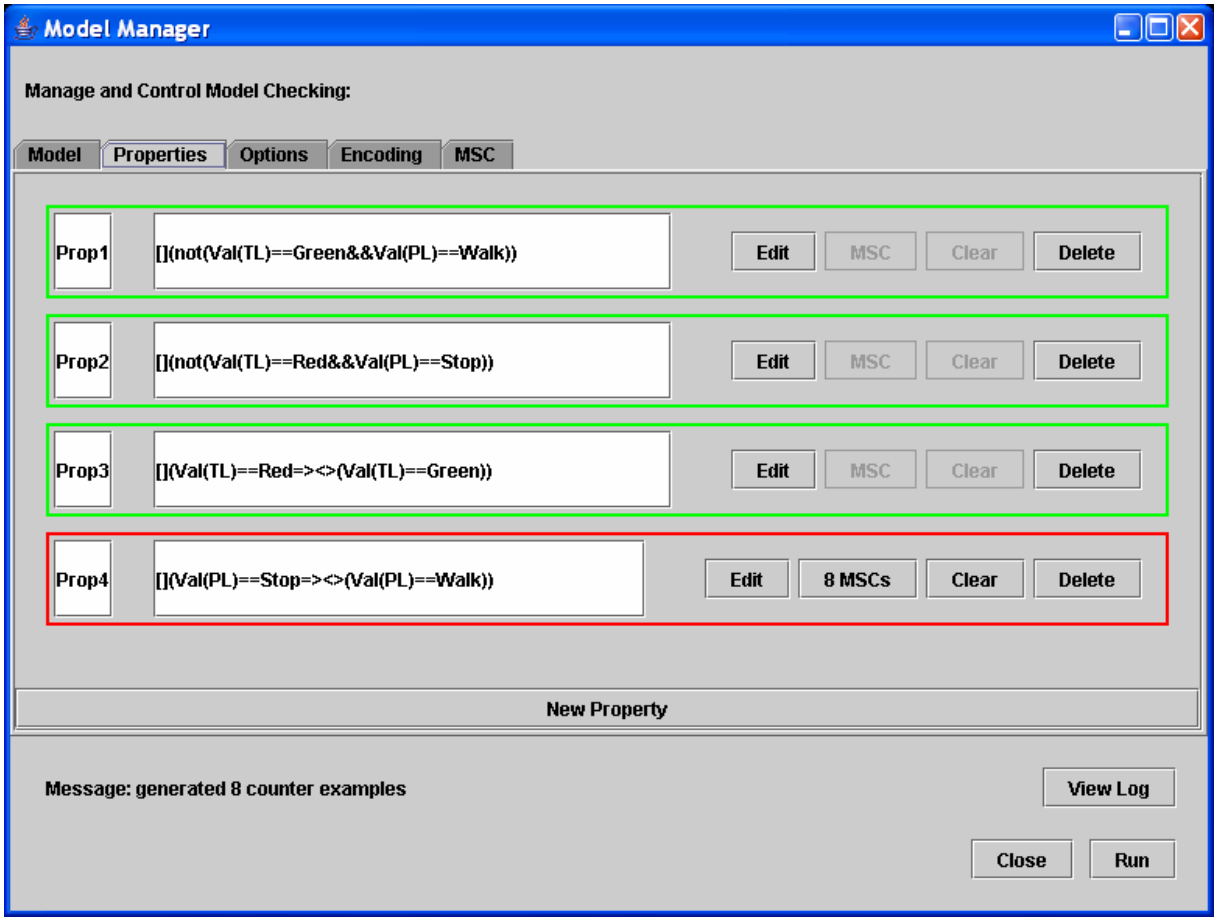

Figure 10: Verification Results

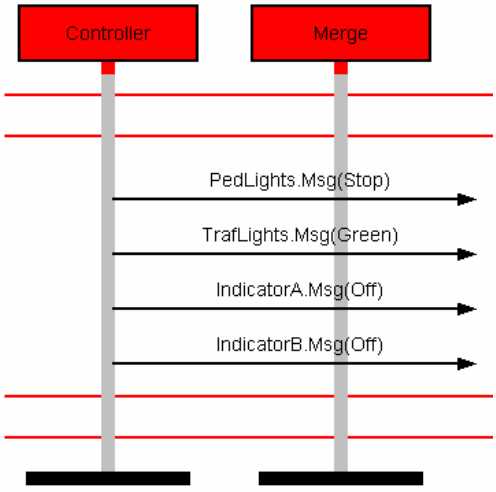

Figure 11: Counter Example

5 Available Settings

The available setting can be chosen from a tabbed pane, which has the following panels:

- Model: for the selection of the checked model (see Section 5.1)
- Properties: for the definition of properties and the results (see Section 5.2).
- Options: for additional options of the model checker (see Section 5.3).
- Encoding: for the encoding of the model into the model checker (see Section 5.4).
- MSC: for the configuration of the generated counter examples (see Section 5.5).

The settings are stored in the file **`validas.cfg`** in the working directory of AutoFOCUS and will be used when MCUI is invoked the next time.

5.1 Model

In the Model panel the root component for the verification can be selected. The default root component is the top component in the project, or the component which has been selected when starting MCUI (in AutoFOCUS as well as in AutoFOCUS 2).

The root component can be selected by pressing the Browse button as depicted in Figure 12.

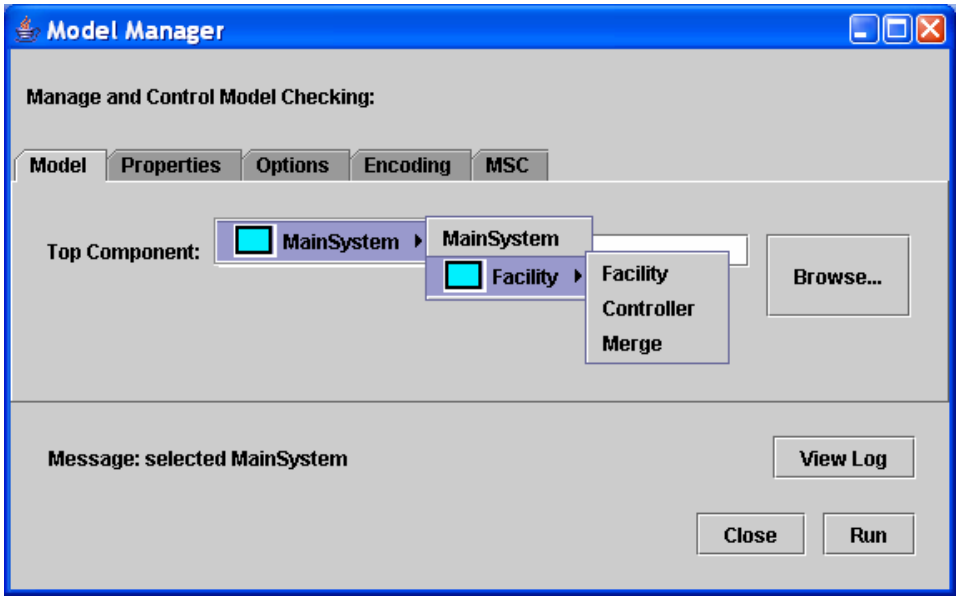

Figure 12: Model Panel

5.2 Properties

In the Properties panel the properties of the selected component can be defined for verification and the results are displayed. Figure 13 shows the properties specified for the example in Section 4.1.

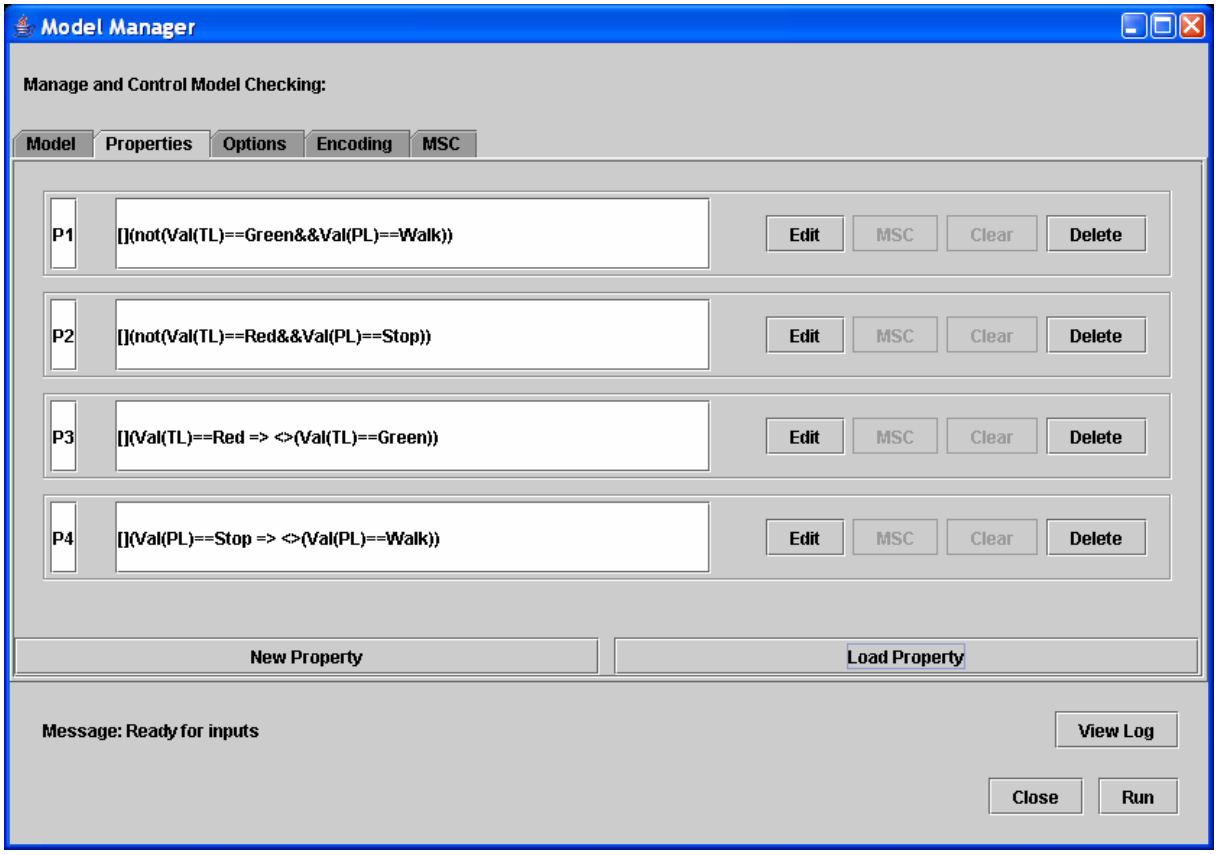

Figure 13: Properties Panel

The Properties displays the properties of the selected model. It supports the addition of new properties to the selected component using the New Property button. With the Load Property button properties can be loaded from files. Every property has a name, that can be changed and an expression. By pressing the Edit button the property editor starts (see Figure 14) for editing the properties. The property editor supports the insertion of all user defined functions, constants, ports, states and variables. Properties can also be saved to files in the editor. After closing the editor with save & close the property has changed.

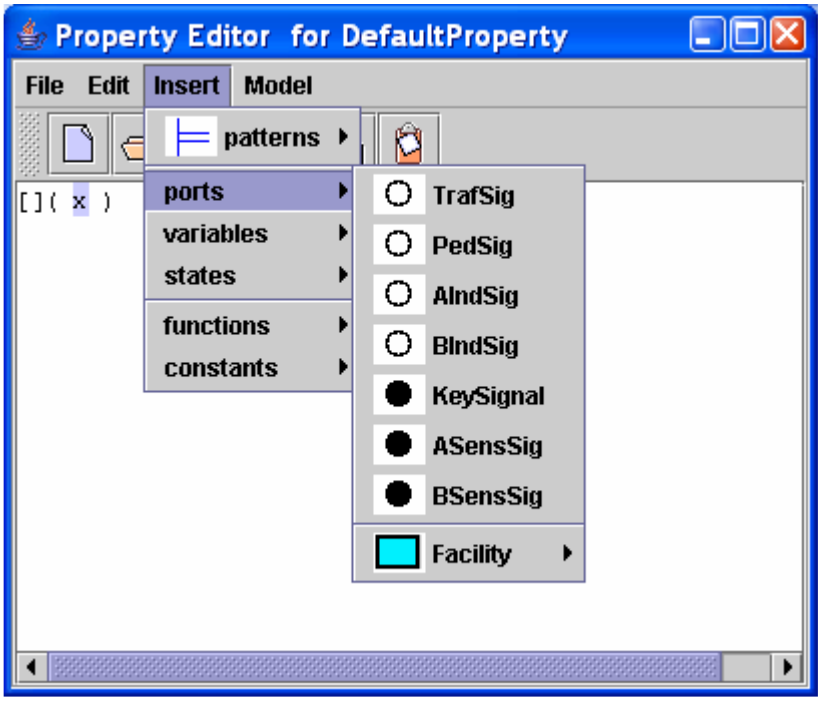

Figure 14: Property Editor

After the verification of the properties the verification result is displayed graphically: red properties are invalid, green properties are valid (see Figure 10 on page 15). If the verification fails it can have the following reasons:

- Property is syntactically not correct, e.g. `[](x>)`,
- Property is not type correct e.g. `[](True>1)`,
- Property contains unbound variables, e.g. `[](X==True)` and
- Property is semantically invalid.

For the first cases only a status messages is issued, in the last cases counter example show the violation of the property. Counter examples can be viewed by pressing the MSC button. Note that in this case all generated counter examples are opened at once. The generation of counter examples can be configured (see Section 5.5).
By pressing the Clear button, or by changing the property the generated counter examples are removed and the validation colour is cleared.

5.3 Options

The Options panel describes general options of the model checking process (see Figure 15).
The following options can be configured:

- Path to checker: can be selected by pressing the Browse button.
- Kind of checker: (NuSMV or Cadence SMV): can be selected using the toggle button
- Additional options for the checker: can be typed directly into the field. The description of the available options can be found in the documentation of the model checkers, e.g. in [NuSMV].
- The maximal runtime in seconds. After this time the model checker is aborted. Model Checking can also be aborted earlier by pressing the cancel button on the verification thread. The Maximal time [s] is also used for the progress bar.
- Bounded Model Checking (see Section 3.4), can be enabled and disabled. If BMC is enabled the maximal bound will be used from the textfield, and properties can not be completely verified, but falsified. The selection of additional verification tools (like chaff) has to be done during the installation. The options for using these can be found in the model checkers manual and can be entered directly in the Additional options textfield, or they can be entered in the configuration file **`validas.cfg`** by specifying the variable `validas.mc.bmc.opt.nusmv` or `validas.mc.bmc.opt.cadence`.

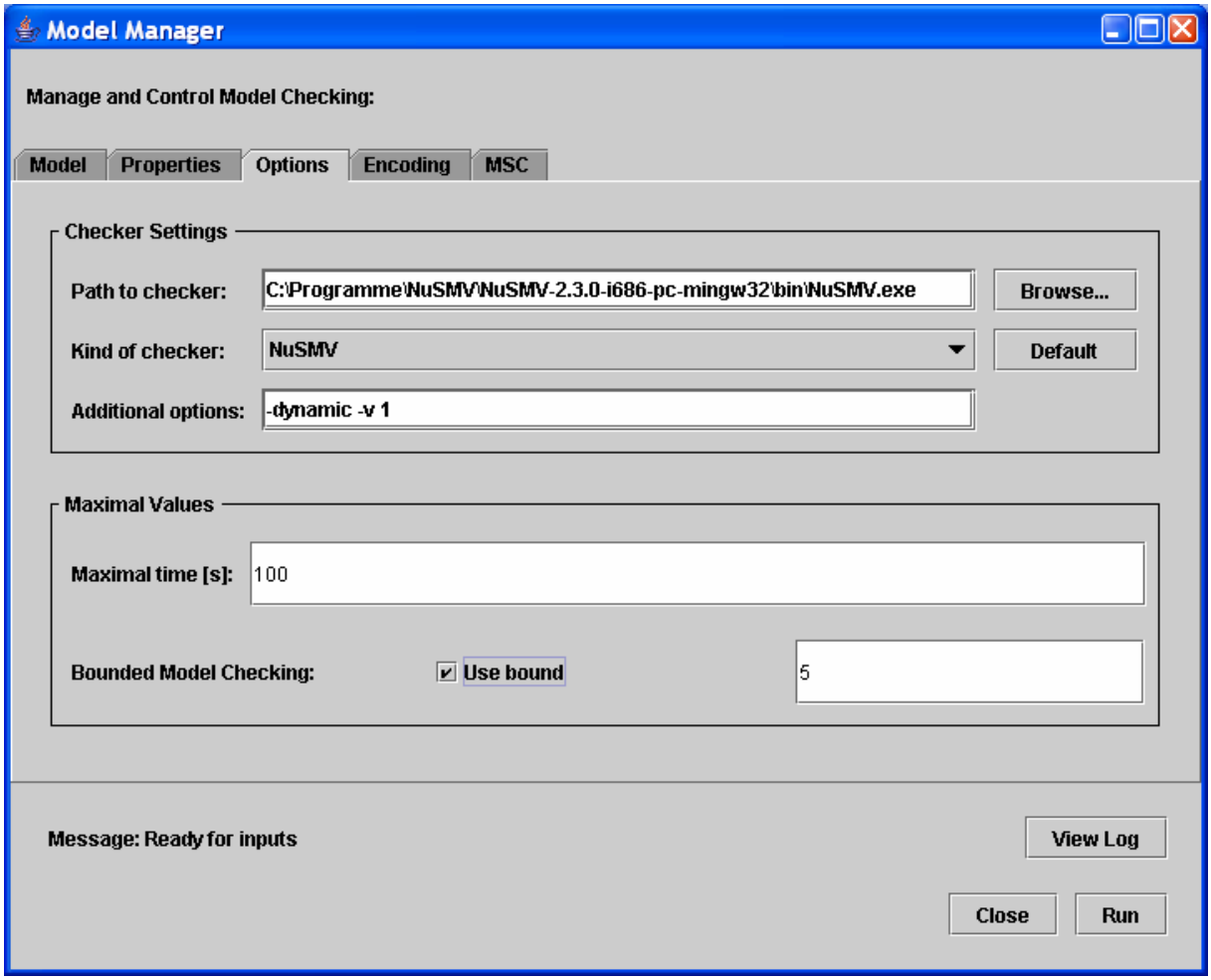

Figure 15: Options Panel

5.4 Encoding

The Encoding panel controls the encoding of the models into the checker language. Depending on the characteristics of the model different encodings are meaningful. They can be configured within this panel (see Figure 16).

In general the results of the encoding can be seen in the generated files, which by default are in the directory **`SMVfiles`** and are named like the selected top component. E.g. **`SMVfiles/MainSystem.smv`**.

The following encoding settings can be configured:

- Use bit arrays in SMV code: is a feature of SMV to store the complex data types more efficiently

- Use modules in SMV code: is a feature to describe the system in a modular way. Every component in AutoFOCUS will correspond to one SMV module
- Encoding of defined data types: allows different encoding of user defined data types. Changing this representation of the data types will make the access to the values of the data types faster (bit field) or reduce the size of the encoding (enumerated)
- Property patterns are a flexible way for defining new properties, for example `never(p) = $AG(not(p))`. Property patterns can be extended by the user by enabling them, which enables the Edit pattern definition button for editing the patterns.
- Intruder properties are a special method for defining hostile attacks to the security of the models (see [Wim05]).
- The Maximal integer value can be configured. This allows many model that use (in AutoFOCUS infinite) integer variables to be checked efficiently. The maximal integer should be not greater (but not less) than the maximal required value.

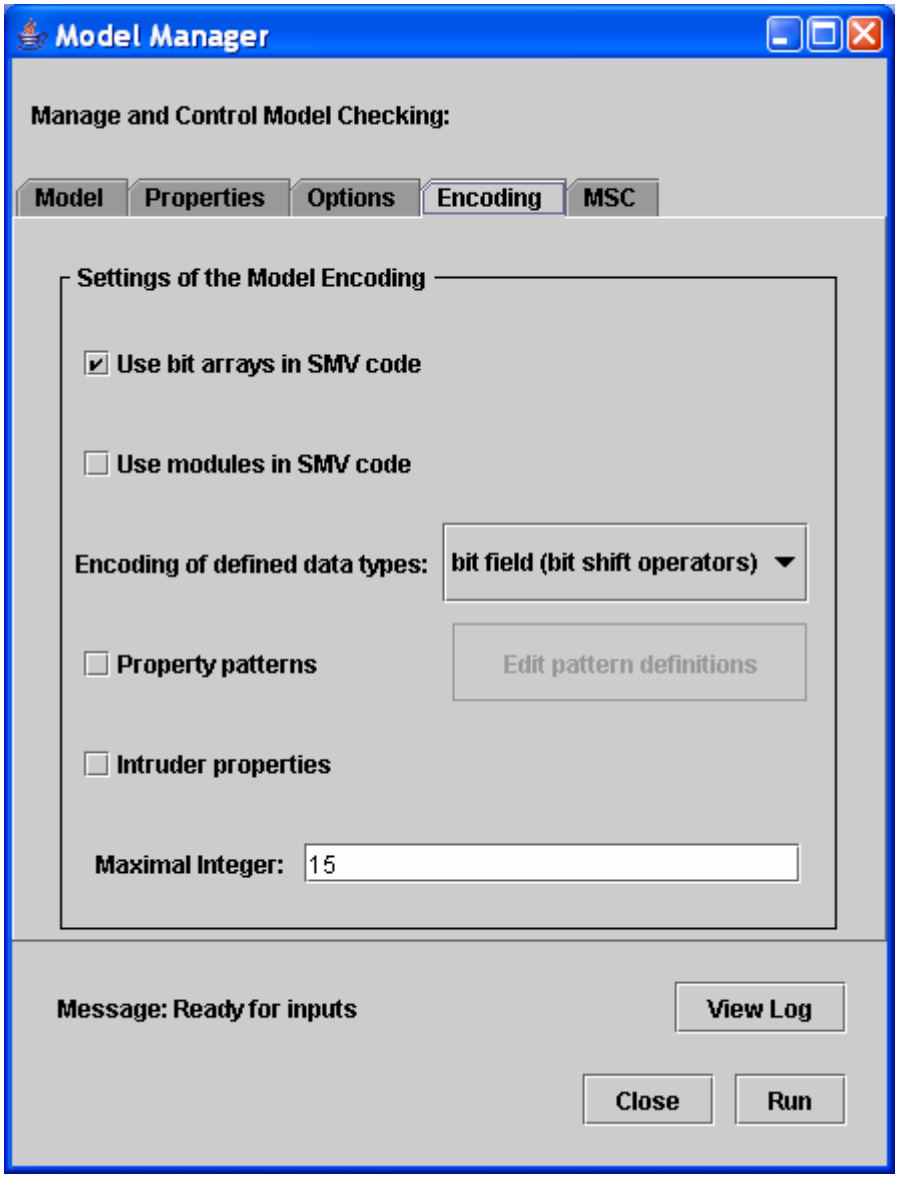

Figure 16: Encoding Panel

5.5 MSC

The MSC panel allows to configure the generation of counter examples (see Figure 17). The following options can be set:

- Generation of MSCs can be controlled (note that in some cases identical MSCs can be generated, e.g. if only one component is checked)
  - Black-box MSC: is a MSC that shows the selected component as a black box (only from outside)

  - Component MSCs: for every verified component a black-box MSC and for every component with a structure a MSC with internal messages are generated if this option is chosen
  - Basic MSC is a MSC with all atomic components in the verified system
- Generation of MSC elements allows to select the generated elements in the counter examples. State and local variable information can be added to MSC in all steps, or only if the values change.

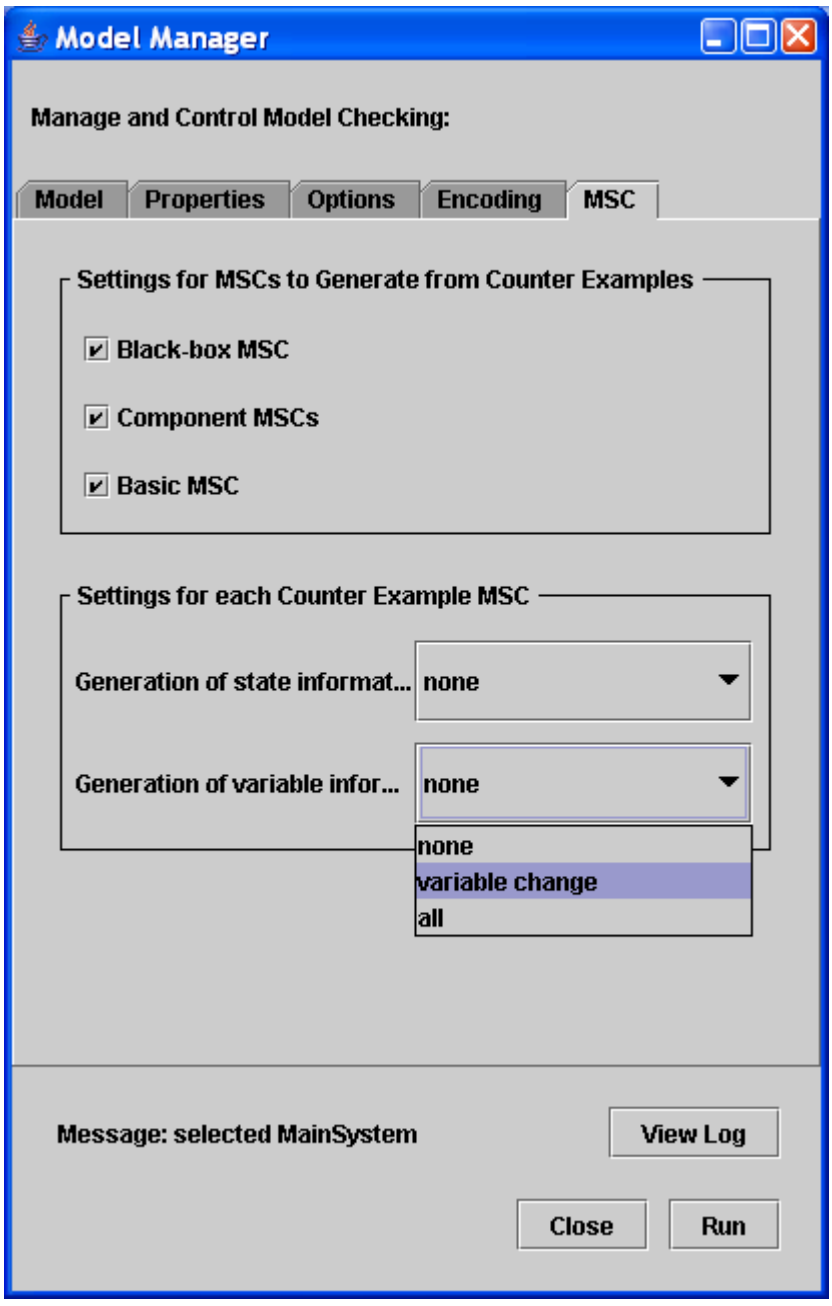

Figure 17: MSC Panel

6 Known Bugs

The following bugs are known in MCUI and the AutoFOCUS model checker:

- MSC editor does not depict MSCs nicely, especially loops are not depicted.
- It is recommended to restart AutoFOCUS before MCUI is started, especially if the model has been changed.
- The SMV subset check (see Figure 2 on page 7) does not locate the positions of the errors, so it is the users task to locate and remove the un-checkable elements.
- With Cadence SMV, only one LTL property can be checked at the same time. (if more than one property is specified, they are conjugated and association of property to counterexample does not work) If there is no counterexample, the corresponding property is not marked yellow. (Cadence SMV only writes an empty .out-file)
- Qualification of Ports does not always work (e.g. crash when using SystemEnv.TrafSig in properties for TrafficLightsControll)
- Tagged properties {sysproperties=…} must not contain property patterns (e.g. "never(…)").
- Syntax errors in properties should be reported more explicitly (currently, only via the console)
- MCUI might crash if model is not simulatable (e.g. no initial state in an automaton)

7 Literature

The following publications are available for the model checking of AutoFOCUS (most of them can be downloaded from the technical university of Munich):

[CGH97] E. Clarke, O. Grumberg and K. Hamaguchi, Another Look at LTL Model Checking. Formal Methods in System Design, 10(1):57-71, February 1997.

[NuSMV] NuSMV User Manual

[PS99] The Quest for Correct Systems: Model Checking of Diagramms and Datatypes, Proceedings of Asia Pacific Software Engineering Conference 1999, 449-458, Philipps, Slotosch

[Wim00] Specification Based Determination of Test Sequences in Embedded Systems, Guido Wimmel, Diploma Thesis, Technische Universität München, 2000

[Wim05] Model-Based Development of Security-Critical Systems Guido Wimmel, PhD thesis, TU München, 2005

[Slo00] Modelling and Validation: AutoFocus and Quest, Oscar Slotosch, Formal Aspects of Computing, 2000 12:225-227